Creating Permanent Test Collections of Web Pages for Information Extraction Research*

Bernhard Pollak and Wolfgang Gatterbauer

Database and Artificial Intelligence Group
Vienna University of Technology, Austria
{pollak, gatter}@dbai.tuwien.ac.at

Abstract. In the research area of automatic web information extraction, there is a need for permanent and annotated web page collections enabling objective performance evaluation of different algorithms. Currently, researchers are suffering from the absence of such representative and contemporary test collections, especially on web tables. At the same time, creating your own sharable web page collections is not trivial nowadays because of the dynamic and diverse nature of modern web technologies employed to create often short-lived online content. In this paper, we cover the problem of creating static representations of web pages in order to build sharable ground truth test sets. We explain the principal difficulties of the problem, discuss possible approaches and introduce our solution: WebPageDump, a Firefox extension capable of saving web pages exactly as they are rendered online. Finally, we benchmark our system with current alternatives using an innovative automatic method based on image snapshots.

Keywords: saving web pages, web information extraction, test data, Firefox, web table ground truth, performance evaluation

1 Introduction

In the visions of a future Semantic Web, agents will crawl the web for information related to a given task. With the current web lacking semantic annotation, researchers are working on automatic information extraction systems that allow transforming heterogonous and semi-structured information into structured databases that can be later queried for data analysis. For testing purposes researchers need representative and annotated ground truth test data sets in order to benchmark different extraction algorithms against each other. Whereas such ground truth data sets exist in the image recognition domain (which are sometimes also used for table extraction research like the University of Washington document image database III [16] which contains up to 215 marked tables) this is not the case for web based table extraction. The difficulties

* This work has been supported in part by the Austrian Academy of Sciences through a DOC scholarship, and by the Austrian Federal Ministry for Transport, Innovation and Technology under the FIT-IT contract FFG 809261.

arise from the fact that information on the web is very volatile and elusive or 'ephemeral', as researchers in the web archiving community call it.

It was estimated that web pages disappear at a rate of 0.25-0.5% per week [4] or ~12-23% per year, and that about half out of all pages that are still available after one year occur at least minor changes [15]. This means that during the time between researcher *A* performing research, submitting a paper to a conference, actually presenting the results, and researcher *B* evaluating his algorithms on the same test set as specified by the referenced URL list, a considerable amount of the test web pages are already gone or changed from the previous evaluated version.

Current approaches like the "save complete" function of web browsers and available web downloader's like HTTrack from the major web archiving initiatives currently do not address this issue in a satisfactory manner. Faced with the problem in our project no available tools fully satisfied our needs.

2 Related Work

The lack of standard data sets in the table extraction field is shortly noted by Hurst [9]. Also Wang and Hu [17] observed the absence of public available web table ground truth sets and were, to our knowledge, the first who made their annotated web table test data available for downloading. However, their test collection was created around 5 years ago and includes only HTML code.

A considerable number of initiatives and literature exist on the issue of web archiving [3]. Notable projects are especially the San Francisco based Internet Archive[1] and the Australian PANDORA System[2]. While focus in this community is predominantly on issues like enabling long-term accessibility, avoiding obsolescence of media, and selection of what to preserve for future generations, these projects face the same technical challenges as we do of actually saving web pages. However, the saving functions of web crawlers and downloader's like HTTrack[3], widely used by web archiving initiatives, do not address important saving problems like such created by dynamic JavaScript as described later. Some web archiving literature [13] mentions such issues as important problems but these difficulties are, to our knowledge, currently not solved by any of these initiatives. The work most related to our problem is the Firefox extension Scrapbook[4], from which we used some of the source code and which we will also describe in more detail in a later chapter.

3 Save Approaches or "Why Saving Web Pages is Hard"

Saving web pages is hard because we have first to answer the question what the actually "correct" web page is. Web pages are implemented with different

[1] `http://www.archive.org`
[2] `http://pandora.nla.gov.au`
[3] `http://www.httrack.com`
[4] `http://amb.vis.ne.jp/mozilla/scrapbook`

technologies like ASP, JSP, PHP and generated dynamically on the fly when a user wants to see the page. So in most cases “the web page” as well-defined entity like a Word or PDF Document does not exist. While originally never intended to generate an exact visual reproduction at every client HTML was extended and new concepts (JavaScript, CSS) were introduced to fulfill more complex visual and interactive requirements. However, a web page is still no exact reproduction and should never be, since the clients have to be robust against different screen resolutions, resizing, installed fonts and the operating system. Depending on the used layout engine (Trident, Gecko, KHTML, Presto,…)[5] the various browsers generate a more or less different rendering of a web page. Not enough web servers will sometimes send different versions of a web page based on the used browser.

The only point where everything comes together is at the screen in front of the user who “measures the correctness” but without any “original” version for comparison. The conclusion is the necessity to register the researcher’s experienced visual representation; thus the used browser together with the web page test repository for interpreting and discussion.

We chose the web browser Mozilla Firefox[6] as our base browser mainly for three reasons: (1) Firefox's extension feature make it easy to adopt functionality to our own needs. (2) It is one of the web browsers most compliant with recommendations of the W3C[7] (for quantitative but controversial evaluation see [6]) and its open source nature and broad acceptance will ensure it to be developed further to handle any future web technologies. (3) Firefox is already widely and successfully used for Semantic Web related applications in the academic literature [2,7,10,11,12].In our opinion, these points outweigh the arguments that internet users still use the web browser Internet Explorer (IE)[8] more often than Firefox (worldwide share of IE ~83% as compared with ~13% of FF in July 2006 according to [14]) and, as a consequence, that there are still a small number of web pages that could not be correctly rendered in other browsers than IE.

The creating of web page repositories has some relations to the web archiving domain but with different requirements (Table 1). We need easy interchangeability, relative small size, a client centric approach and static visual reproduction. There is no need to reproduce dynamic behavior (e.g. JavaScript) which makes this task easier because we could freeze the state of the web browsers DOM tree.

If we want to test a visual based algorithm like VENTrec [5] we have to restore the exact view of the original web page for comparison. To fulfill these offline requirements we analyzed approaches for making a web page locally permanent. At the first step we can distinguish two different levels: the (local) server and the local client.

For the server approach we could use a proxy as transparent layer between the server and the requesting client providing the resources either from the proxy cache or (if not present) from the original server updating the cache for further requests. The primary intention is speed and reliability for the clients. The simple idea for the test

---

[5] http://en.wikipedia.org/wiki/Comparison_of_layout_engines

[6] http://www.mozilla.com/firefox

[7] http://www.w3.org

[8] http://www.microsoft.com/windows/ie

database was to use a proxy to cache the web pages which are then accessed a first time through the proxy resulting in a permanent copy at the proxy server.

On the other side, we have the simple client based approach resulting in a local copy of the web page inside a directory structure which has several advantages and disadvantages depending on the used technique which will be further examined in the next chapter.

Table 1. Our local web page repository objectives vs. web archiving requirements.

	Local web page repository	*Web archiving*
Data size	small	large
Reproduction	static	static and dynamic
Target	single web pages	(parts of) web sites
Interchange	easy	complex
Type of files	visual representation formats (e.g. HTML, CSS; Images)	all formats (e.g. sound, pdf)
Information	special domains	information valuable for future generations
Platform	local client/proxy	high performance internet/intranet servers
Costs	(nearly) none	high

The main disadvantage of a proxy approach is that any one web page is not treated as a single entity. It is nearly impossible and very difficult to identify the files which together generate a specific web page. Normally, proxies use hashed file names and particular directory structures. We found a proxy named SmartCache[9] using original filenames and readable directories, which would theoretically allow better physical localization and therefore tagging but this does not solve the problem because the web page files are often spread about different domains and/or directories leaving the identification of the files difficult. A second problem is to focus the capturing process on well defined web pages. It is difficult to exclude the search web pages and rejected web pages from the resulting proxy data set. So the resulting proxy set may contain unnecessary files and is not minimal.

Table 2. Comparison between local save (the WebPageDump solution) and an alternative proxy approach. Decisive factors for choice of local save are bold.

	Client local save	*Proxy approach*
Tagging/annotation	**no problem**	**very difficult (only separated files)**
Access of files	**no problem**	**very difficult (hash files, directories)**
Interchange	**easy (file package)**	**easy (proxy state package)**
Ease of use	easy	more difficult (set up a proxy)
Package size	minimal	nearly minimal
Same URL repeatedly	possible (versions)	difficult (only separated proxy states)
Server sided logic	no problem	maybe problems (POST, https,…)
Use of “original” files	no (processed files)	nearly
Use different browsers	no	yes
Maintenance effort	high (new standards)	low (doesn’t depend on specific tech.)
Browser transparency	lower	high (same as using orig. URL)

[9] `http://scache.sourceforge.net`

And finally some of the files will be stored but remain not accessible through the web browser because of special server sided authentication logic. For our needs of building web page test collections with additional annotations, ease of use and sharing facility between the researchers, the arguments speak for the local save approach (Table 2).

4 Local Save Solutions

Saving a web page locally could be achieved in several ways. First we analyze simple present solutions which are maybe the first approach for many researchers: the browsers built-in local save functionality and the use of web site downloader's or offline readers. For testing and analyzing we found some web pages which include special web techniques that turned out to be difficult for local saving and served as falsifying test cases (see Table 3).

Table 3. Web page examples.

Web site containing web page	*Properties*
`http://booking.expedia.de/...`	JavaScript generated table and image links
`http://rewies.cnet.com/...`	external CSS files with @import directive
`http://www.mozilla.org/projects/`	image references through CSS
`http://de.selfhtml.org/html/...`	simple frameset and iframe (embedded frame)
`http://complexspiral.com/`	contains a CSS universal selector
`http://www.vor.at/`	frameset with GET variables "Fahrplan"

4.1 Browsers Built-In Local Save

Only browsers which are capable of saving a web page completely (including embedded file types) are investigated further. For correct evaluation of the capabilities it is absolutely necessary to ensure an offline test and clear the browser cache before testing or using different profiles when possible.

The main difference between the browsers beside other specific errors (Table 4) is the handling of JavaScript. While all browsers provide a "View Source" functionality which shows the HTML source as sent by the corresponding web server there are differences when saving the web page "complete". Firefox saves the page with previous executed JavaScript reflecting the actual DOM tree inside the memory. In contrast, Internet Explorer and Opera[10] save the (formatted) HTML code as sent without reflecting JavaScript modification to the HTML code (see also chapter 5.1). Opera does the best job in saving, but has the very serious disadvantage of not saving frames and the confusing behavior of sometimes creating strange "0-Byte" image files.

[10] `http://www.opera.com`

Table 4. Observed principle browser save problems.

Problem Description	*Firefox 1.5*	*IE 6.0*	*Opera 8.53*
Referencing with absolute windows style filepath	-	X	-
CSS files included with the @import are not saved	X	-	-
CSS files from embedded documents are not saved	X	-	-
CSS image references are not saved	X	X	-
CSS universal selector is removed	-	X	-
JS dynamic references are not saved	-	X	X
JS Double Execution Problem	X	-	-
Frames are not saved	-	-	X
External iframe ref. from other domains not saved	X	-	-

4.2 Web Site Downloader/Offline Reader

Web site downloader are responsible for downloading web sites from the Internet to a local directory, building recursively all directories, getting HTML, images and other files from the server. Normally, they would provide better results than the simple local save functionality of the browsers above.

The main problem is the JavaScript processing. Such software does not generate a DOM tree for visualization but rather uses parsing techniques to determine the files for downloading and will fail including dynamic JavaScript generated content, like the travel agency logos or the dynamic online dependent menu from the Expedia page (Table 3). This is the conceptual reason why web site downloader's will make errors when downloading a web site. To overcome this limitation there is no alternative to direct DOM processing which presents the visual result of JavaScript code.

Another problem is that this kind of software leads to an overkill. Normally, the web site downloader does not distinguish between external iframe or frame references and "normal" link references. In order to get a working copy of a single web page it is necessary to increase the local and probably the external depth which results in fetching many unnecessary files (especially with extensive web pages including content from many sources). The effort to fetch the complete CNet page (Table 3) with HTTrack results in 20 MB as compared to 300KB with WebPageDump.

The last problem occurs in the used directory structure. Typically, there is a separate directory for every domain created which leads to a complex structure. Admittedly HTTrack has the ability to save flat with random filenames.

Potentially available configuration options for reducing the amount of files have to be adapted to each site. These disadvantages make this approach totally unpractical especially for the easy interchange between researchers and regarding a standardized directory/file naming.

5 Saving Problems

5.1 The JavaScript Double Execution Problem

Consider the following simple example:

```
<script type="text/javascript">
  function WriteHello () {
    document.write('Hello ');
  }
  WriteHello();
  </script>
  <p>End</p>
```

which results in displaying “Hello End”. Saving the web page with Firefox (Web Page, complete) and reopening gives: “Hello Hello End”.

This behavior is caused through ‘double execution’ of the JavaScript code. When the page is loaded JavaScript is executed resulting in an HTML insert of “Hello”. When saving the page the code is saved in the present state (meaning together with the inserted HTML code). Reopening the file results in a second JS execution inserting another “Hello” giving two “Hello” entries.

At Expedia the JavaScript code generates a dynamic table and a dynamic online dependent menu. The saving and reopening with Firefox results in two menus and table row doubling. This would be unacceptable especial for the table extraction domain.

On the other hand, the Internet Explorer saves the HTML code as sent without JavaScript code executed, thus avoiding the JS double execution problem. Expedia shows that this approach is also problematic since the JavaScript code inserts dynamic URL references for travel agency logos. When saving with the IE, these images are not included, resulting in an incomplete local copy.

WebPageDump removes JavaScript code because this is the only possibility to avoid dependences and side-effects. All layout related changes are reflected inside the DOM tree after the JavaScript execution.

5.2 The Character Set/HTML Entity Problem

Firefox depends heavily on the DOM tree generated inside the memory, so there is no possibility to process the code as sent from the web server (without JavaScript executed). The saved HTML source code is always JavaScript processed which avoids the JS double execution problem explained before.

But this independence introduces another problem; the so called HTML entities could not be reproduced. The DOM tree of Firefox is based on the UTF-16 Unicode format and converts all HTML entities to the corresponding chars. There is no possibility to go the way back. If a specific character set is used (e.g. Shift_JIS), entities like `&middot;` are converted to an UTF-16 char. If we convert this char code directly back to the original charset we receive undefined chars displayed as question marks inside the web page. The simplest solution is to convert not to the original charset but to UTF-8, which is functional equal to UTF-16 except some space issues.

Unfortunately this could change the used font depending on which font is available for which charset.

WebPageDump uses the original intended charset as output and detects if all chars are presentable in the final charset avoiding imprecise font rendering. Possible HTML entities are detected and converted to their correct textual representation regardless of their original format (either direct char or text entity) in the original page. For entities which are not known by Firefox a numbered entity is used. However, if the web page contains an erroneous charset definition WebPageDump will probably fail because the chars are then not correctly mapped into the Unicode range and we cannot restore the original char codes. Addressing this problem would require a change of the JavaScript/DOM Specification towards an additional DOM Node attribute which contains the original raw text from the HTML file.

5.3 Rendering Bugs

There exist some strange layout dependencies inside Firefox which were detected through the image based testing and had to be circumvented. For example the layout changes depending on the position of the DOCTYPE definition. Another example is the "src" Attribute of the `<script>` tag. Even an empty src attribute could change the layout compared to the functionally equivalent removing of the whole `<script>` tag.

But WebPageDump can not resolve all of this bugs. They have to be addressed by the Mozilla/Firefox development team itself. This is the reason why 100% correctness is not possible through WebPageDump.

6 The WebPageDump Solution

On the search for an existing solution to our problem we found the Firefox extension Scrapbook, whose development started in December 2004 and was available in version 1.0 when we began our project in April 2006. Scrapbook enables the organization of information collected from the web in a tree sidebar. Because there is much functionality we don't need we extracted and restructured only the relevant saving part and adapted the software to our needs:

Quality Assurance. Although Scrapbook does a good job in saving a web page, but is not intended to make a really perfect local copy of a web page. Especially the character code handling is unaccounted. This may result in a different font rendering as stated above. Also the HTML Entities and special rendering are not treated in any way.

Introducing command line functionality. As result the software can be used from external research tools and for automating the testing process. This introduces a single oriented mode and two different batch modes, one for URL lists and one for existing local web page collections. Also a special command-line flag enabling to continue an interrupted batch process was implemented.

Test suite. WebPageDump was extended to serve as a test suite. We integrated the control of the Pearl Crescent Page Saver[11] Firefox extension, which is capable of saving an image of the whole web page, so we could use this functionality directly from inside WebPageDump especially within the batch modes. This made it possible to automatically check a great amount of web pages which is a necessary precondition for improving the quality and make comparisons. Also an evaluation mode for checking the resulting images including the generation of simple test reports was implemented.

Automatic directory naming scheme. We wanted to support the researchers notably from the table extraction domain in easy generating a web page collection. For this reason we developed an automatic naming scheme when saving a web page to a target directory. This WPD naming specification (explained in detail below) focus on easy readable directory names and the possibility for detecting already saved URLs together with included version information in the directory name itself. As a result the researcher need not to worry about the naming issues. He selects a target directory and saves all intended web pages inside this directory and the naming is done completely automatic, regardless of saving the same URL more than once.

6.1 The WPD Naming Specification

For a relative short readable directory name we use the domain name and a modulo 10000 counter adding up the ASCII codes of all chars including possible GET variables. This method leaves a small possibility of double names. If this is the case WebPageDump introduces a counter separated by "c" which is added to the directory. If we want to store the same address (due to content/layout changes) a counter separated by a dot is added indicating the web page version (e.g. `www_cnet_com_0003.1` or `www_cnet_com_0003c1.1`). The first web page has always ".0" added meaning the first saved version.

WebPageDump uses a flat directory design which is derived from Scrapbook leaving potential subdirectories for additional purposes (e.g. testing, annotations,…). "index" is used as naming scheme for the HTML and CSS files including a counter (`index.html`, `index_1.html`,…). All other files are saved using the original name (with a counter added for double names.

Inside the HTML Files WebPageDump stores meta tag information about the used version, the original base URL, the specific file URL and the current date/time, so researchers can reconstruct where the web page was originally located and when the web page was added to the collection.

```
<meta name="wpd_version" content="1.0">
<meta name="wpd_baseurl" content="http://forum.tatar.info/">
<meta name="wpd_url" content="http://forum.tatar.info/">
<meta name="wpd_date" content="2006-8-28T9:24Z">
```

This WPD Specification for naming could serve as a simple standard naming for the base research where mostly small web page collections are generated for testing.

---

[11] `http://pearlcrescent.com/products/pagesaver`

7 Automatic Visual Based Evaluation

A serious problem in our first tests was the time between selecting the URLs and running the WebPageDump tests which results in unavailable web pages. So we decided to use the SmartCache proxy specialized for offline browsing (see Save Approaches above). A proxy is ideal for testing the visual differences between the local and online version because we don't need direct access to the proxy files and we don't want to store different versions of the same URL. Also we avoid websites which may need special server handling because we are primary interested in complex charset/layout issues.

We accessed the web pages through the proxy and wrote the URL to a text file. This text file was the input for the WebPageDump batch mode which saved every web page and called the PageSaver Extension afterwards resulting in an image snapshot of the original first web page view inside the browser. Then a second run was done with another WebPageDump batch mode using the local saved web pages as a source for the image generation. Because of the compression features of the used PNG image format we could compare the file size of the different images. It would be quite improbable that different images will result in the same file size. This would not be the case with an uncompressed image format where only the image dimension will determine the file size.

The problem of dynamic animated content for the comparison (particularly Flash, Movie and GIF files) was addressed by blocking the files with the Adblock[12] extension (blocked extensions: swf, svg, mms://, rm, mov, wmv, asx, rpm, wma, wvx; GIF Animations were deactivated by WebPageDump itself).

To achieve a high degree of generality, we used random URLs, a manual language selection and a selection based on our VENTrec approach which used web pages from the digital camera domain. The random data set was generated with the Mangle Random Link Generator[13] providing an interface to Google through selecting random words from a word database. The results are mostly web pages in English (Mangle Dataset). The other language based URLs were selected manually with the Google directory starting from the "world" entry. From every language we selected two random web pages (Languages Dataset). Thereafter we focused on three additional Languages (also from the Google directory): Chinese, Japanese and Arabic because of the difficulties especially for the right to left text direction in Arabic and the sometimes "exotic" layout/view of the Japanese web pages. Chinese was selected because it is the most spoken language on the world (ignoring the dialects). The last data set was generated with the VENTrec approach in mind searching for the first 100 Google results with the keyword: "Canon Digital IXUS 800 IS" (Digicam Dataset).

We calculated the correctness taking the byte differences from the screenshots and benchmarked our approach with Scrapbook and the Firefox "save complete" function. Table 5 shows that WebPageDump performs significantly better than Scrapbook and Firefox. We achieved a correctness of 91.8% as compared to 64.7% and 36.9%. The ">100" column ignores the variations of less than 100 Bytes which could be caused by animations and not necessarily indicate errors. Due to different bugs of the Firefox

[12] `http://adblock.mozdev.org`
[13] `http://www.mangle.ca`

implementation there is a border for additional improvements and we estimate that we cannot increase the correctness much more than the actual results.

Table 5. Correctness benchmark of WebPageDump with Scrapbook and Firefox.

Category	*Count*	*WebPageDump*		*Scrapbook*		*Firefox*	
		correct	*>100*	*correct*	*>100*	*correct*	*>100*
Arabic	50	84.0%	92.0%	56.0%	68.0%	32.0%	34.0%
Chinese	50	86.0%	88.0%	54.0%	62.0%	34.0%	38.0%
Digicam	99	86.9%	92.9%	48.5%	66.7%	15.2%	19.2%
Japanese	49	91.8%	91.8%	73.5%	87.8%	44.9%	53.1%
Mangle	50	94.0%	96.0%	66.0%	76.0%	44.0%	44.0%
Languages	152	98.7%	100.0%	78.3%	88.2%	48.7%	55.3%
Sum	450	**91.8%**	94.9%	**64.7%**	76.9%	**36.9%**	41.6%

8 Conclusions and Outlook

Information extraction research needs representative and annotated ground truth test sets in order to benchmark different extraction algorithms against each other. In addition to the existing difficulties of deciding on the actual ground truth, web information extraction also needs tools to make web pages permanent. Current approaches like the "save complete" function of web browsers and available web downloaders like HTTrack from the major web archiving initiatives currently do not address this issue in a satisfactory manner.

To overcome these limitations, we developed the Firefox extension WebPageDump which builds upon the existing work from the related initiative Scrapbook and extends this software to introduce really perfect local copies (considering HTML Entities and different Firefox Bugs), together with a quality assurance concept (command line support, batch modes, including the PageSaver extension) which is the precondition to achieve the goal of a high visual concordance of the local copies. Also the researchers requirements are considered through the automatic management of the directory naming and versioning issues.

WebPageDump takes a browser centric view instead of a document centric view towards web information extraction. As such, it has nearly no conceptual limits (in contrast to website downloader's) for saving static web pages as it only depends on the DOM tree which corresponds directly to the visual representation including possible dynamic changes apart from some bugs and the absence of raw text access.

The results show that the WebPageDump performs much better than the Scrapbook Extension and of course the internal "local save" function of browsers. Our next focus will be to implement our conceived annotation methodology in order to create a representative web table ground truth. As well, the extension together with the evaluation test set will be put online[14].

[14] `http://www.dbai.tuwien.ac.at/user/pollak/webpagedump`

Acknowledgments. We would like to thank Prof. Georg Gottlob for overall motivation of this research, Wolfgang Holzinger and Bernhard Krüpl for helpful discussions in the course of this work, and one of the anonymous reviewers for helpful comments.

References

1. P. Bailey, D. Hawking, and A. Krumpholz. Toward meaningful test collections for information integration benchmarking. In *Proc. IIWeb at 15th WWW*, May 2006.
2. J. Carme, M. Ceresna, O. Frölich, G. Gottlob, T. Hassan, M. Herzog, W. Holzinger, and B. Krüpl. The Lixto Project: Exploring New Frontiers of Web Data Extraction. In *Proc. 23rd BNCOD*, Springer. July 2006.
3. M. Day. Preserving the Fabric of Our Lives: A Survey of Web Preservation Initiatives. In *Proc. 7th ECDL*, pp. 461–472, Springer. August 2003.
4. D. Fetterly, M. Manasse, M. Najork, and J. L. Wiener. A large-scale study of the evolution of web pages. In *Proc. 12th WWW*, pp. 669–678, ACM. May 2003.
5. W. Gatterbauer and P. Bohunsky. Table Extraction Using Spatial Reasoning on the CSS2 Visual Box Model. In *Proc. 21st AAAI*, pp. 1313–1318, AAAI/MIT Press. July 2006.
6. D. Hammond. Web browser standards support summary. 2006. Available: `http://www.webdevout.net/browser_support_summary.php` (August 2006)
7. W. Holzinger, B. Krüpl, M. Herzog. Using Ontologies for Extracting Product Features from Web Pages. In *Proc. 5th ISWC*, Springer. November 2006.
8. J. Hu, R. Kashi, D. Lopresti, G. Nagy, and G. Wilfong. Why table ground-truthing is hard. In *Proc. 6th ICDAR*, pp. 129–133, IEEE. September 2001.
9. M. Hurst. Layout and language: Challenges for table understanding on the Web. In *Proc. 1st WDA at 6th ICDAR*, pp. 27–30. September 2001.
10. D. Huynh, S. Mazzocchi, and D. Karger. Piggy Bank: Experience the Semantic Web Inside Your Web Browser. In *Proc. 4th ISWC*, pp. 413–430, Springer. November 2005.
11. N. Kannan and T. Hussain. Live URLs: breathing life into URLs. In *Poster Proc. 15th WWW*, pp. 879–880, ACM. May 2006.
12. A. Kerne, E. Koh, B. Dworaczyk, J.M. Mistrot, H. Choi, S.M. Smith, R. Graeber, D. Caruso, A. Webb, R. Hill, and J. Albea. combinFormation: a Mixed-Initiative System for Representing Collections as Compositions of Image and Text Surrogates. In *Proc. 6th JCDL*, pp. 11–20, ACM/IEEE. June 2006.
13. J. Marill, A. Boyko, and M. Ashenfelder. Web Harvesting Survey:Version 1. International Internet Preservation Consortium & The Library of Congress, July 2004.
14. OneStat Press Box. Global usage share Mozilla Firefox has increased according to OneStat.com., July 9, 2006. Available: `http://www.onestat.com/html/aboutus_pressbox44-mozilla-firefox-has-slightly-increased.html` (August 2006)
15. A. Ntoulas, J. Cho, and C. Olston. What's new on the web? The evolution of the Web from a search engine perspective. In *Proc. 13th WWW*, pp. 1–12, ACM. May 2004.
16. I. Phillips. Users' Reference Manual, CD-ROM, *UW-III Document Image Database-III*, 1995.
17. Y. Wang and J. Hu. A machine learning based approach for table detection on the Web. In *Proc. 11th WWW*, pp. 242–250, ACM. May 2002.